ESD-TR-70-408

MTR-2004, Rev. 1

THE VENUS MULTIPROGRAMMING SYSTEM YEAR END SUMMARY

B. H. Liskov

JANUARY 1971

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 700A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract F19(628)-68-C-0365

AD717726

ESD-TR-70-408

MTR-2004, Rev. 1

THE VENUS MULTIPROGRAMMING SYSTEM
YEAR END SUMMARY

B. H. Liskov

JANUARY 1971

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 700A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract F19(628)-68-C-0365

FOREWORD

This report was produced by The MITRE Corporation, Bedford, Massachusetts under Project 700A, Contract F19(628)-68-C-0365 and was monitored by Dr. John B. Goodenough, Electronic Systems Division (MCDS). The work was performed 1 September 1969 through 31 August 1970.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

[illegible]

EDMUND P. GAINES, JR., Colonel, USAF
Director, Systems Design & Development
Deputy for Command & Management Systems

ABSTRACT

The Venus Multiprogramming System is an on-line, interactive system in which several users may run concurrently. It is built on the Interdata 3 from a combination of microprograms and software. This document describes the activity of Project 700A, C^3 Computer System Organization, for fiscal year 1970. The work performed consists of the design and implementation of the software part of Venus. The document gives a chronological account of the development of the software, with particular emphasis on the general principles incorporated in the system and the design considerations which led to these principles.

TABLE OF CONTENTS

SECTION I

INTRODUCTION AND SUMMARY

The long-range goal of this project has been to evaluate new techniques for improving the performance and usability of computers for military command control applications. Studies in previous years led to the selection of microprogramming as an important technology which would contribute to the project goals by allowing a user to change the architecture of his computer to match his requirements. During the preceding year, the Venus microprogram was designed, implemented, and installed on the Interdata 3 computer at MITRE. This microprogram altered the Interdata in significant ways (1, 2).

The work program for 1970 has been to exploit the new machine architecture in order to observe its effect on both the design and production of software. This has been accomplished through the development of the Venus Multiprogramming System, a multi-user, on-line operating system, which is organized around the microprogrammed functions for managing multiple tasks and other new machine operations.

In addition, the Venus Multiprogramming System is designed to satisfy a continuing goal of the project: to ease the problem of program production, in particular the production of large systems of programs. The solution to this problem is provided by the collaboration of the microprogram and the software. The software makes all the features of the microprogram available to the user; among these are the use of virtual memories, which free the user from worrying about storage allocation, and the standard procedure interface and consequent emphasis on procedures, which makes programming logically simpler. The software frees the user from concerning himself with the idiosynchrasies of the devices by providing input/output functions; it provides functions which help the users share data and communicate with each other. Also the software provides debugging aids so that checkout time is reduced.

The acquisition of additional peripheral equipment has increased the ability of the facility to support current and future applications. In conjunction with hardware acquisition and maintenance, the design and implementation of equipment checkout programs has been a necessary activity.

Throughout the year, information about microprogramming developments was actively collected, and the accomplishments of the project were presented at two microprogramming workshops, in Phoenix, Arizona, and in Grenoble, France.

SECTION II

SOFTWARE

APPROACH

The installation of the Venus microprogram transformed the Interdata computer into a radically different machine. The Interdata as delivered was a standard computer supporting arithmetic and logical operations on data in core memory. The Venus machine supports the following in addition to the standard operations:

(1) Streams. These are named virtual memories which are automatically paged between core and disk. Streams are the primary storage structure in Venus.

(2) Procedures. Procedures are stored in streams. CALL and RETURN instructions transfer control between procedures. Push down stacks, also stored in streams, are used to pass data between procedures.

(3) Interrupts for Debugging. The microprogram recognizes the occurrence of certain conditions; when a condition occurs which is enabled, the microprogram provides an interrupt by performing an associated condition instruction.

(4) Multiprogramming. Sixteen jobs may run concurrently on the Venus machine; the microprogram controls the allocation of processor time to the jobs. Semaphores, first proposed by Professor Dijkstra (3), are provided to control synchronization of the jobs and sharing of resources. The microprogrammed multiplex Input/Output channel also uses semaphores to synchronize with jobs.

The reader should refer to documents (1) and (2) for more detailed information about the Venus machine.

Very few software tools were available for use on the Venus machine. The MI-3 Assembler, written for an earlier microprogrammed version of the Interdata, had been modified to assemble the new Venus instructions correctly but did not support the Venus concepts such as the use of streams. Ultimately, we wanted to produce a sophisticated system which itself used many of the new features introduced by Venus and, at the same time, presented an environment to users which would facilitate the design, implementation, maintenance, and execution of large computer-based systems. Clearly there was a large gap between the bare Venus machine with almost no software and the desired end result.

For this reason, it was decided to implement the Venus Multiprogramming System in two stages. The first stage was to provide a single-user environment in which the essential tools would be available for programmers to code and check out programs using the Venus concepts, while allowing only one user at a time to run on the machine. This interim system would then be used to implement the final system, which would support multiple users and have expanded capabilities.

The major components of the interim system were themselves designed to be easily incorporated into the final system, either by modification or incremental addition of code as new capabilities became available. For example, the multi-user system still must provide a running environment for each user; the single-user environment should be close to this environment.

One of the first activities in designing the new software was the establishment of coding conventicns. It is essential to any programming activity which involves more than one person that standard conventions be established, especially with respect to the interfaces between procedures. It is also apparent that the Venus concept of procedures and the machine features such as stacks and interrupts have an important influence on the kinds of conventions that should be established. These conventions were designed to make allowance for future extensions to the system, such as debugging routines to provide traces of procedures and their arguments, and the development of higher level languages in which system procedures might be referenced.

THE SINGLE-USER ENVIRONMENT

Two features were essential to a meaningful single-user environment. First, users needed a way to define and refer to streams (named virtual memories). Second, the user needed a mechanism which would permit him to execute his procedures with and without debugging aids. Building the environment required careful consideration of the basic concepts of the Venus design. Two significant features had immediate impact: streams and jobs.

Streams

A stream is a contiguous set of 64 thousand bytes, any one of which may be directly accessed by giving the name of the stream and the relative location within the stream. Any given stream, in addition to defining its own name, should be able to reference the names of other streams. To support the use of streams, it was necessary to provide the following capabilities:

1. The ability to assemble procedures and data into streams;

2. The ability to name and reference streams.

The first requirement was satisfied by a simple modification to the MI-3 Assembler which enabled it to assemble procedures and data into streams or core; the choice between stream or core was made by the user and specified by means of a new pseudo-operation. The second requirement is satisfied by the introduction of dictionaries.

The machine recognizes a stream name which is a unique 16-bit integer, referred to as the internal name. However, streams may not be referenced by their internal name symbolically, because internal names are dynamically assigned and hence change from day to day. Therefore, the concept of an external stream name was introduced to reference streams symbolically. A dictionary provides the mapping between external and internal stream names; it is one of the most important data structures used by the Venus Multiprogramming System. As a minimum capability for the initial system, it was determined that two levels of dictionaries were necessary: local dictionaries for grouping the names of streams belonging exclusively to a user, and a single global dictionary containing names of streams which may be referenced by all users.

Jobs

A job in Venus is a program running from an in-core job area which contains its registers and control information. A user always starts running as a single job. To start up a job, only a simple program which initializes its job area and then adds it to the pool of running jobs is necessary. This function is available to the user because, although only one user is running, he is checking out the multi-user system and may need to run other jobs concurrently with his own.

Major Programs in the Single-User Environment

The Loader and the Assembler

In the single-user environment, the Loader is an interactive program which is started up in the user's job area. It performs two main functions, always in response to user commands: it will call the Assembler to assemble the next card deck, and it will execute a procedure specified by the user.

The Assembler and Loader cooperate to define and reference streams--the Assembler by recognizing a definition or reference to an external stream name, and the Loader by using the dictionaries to supply the corresponding internal names. The initial input to the Assembler specifies whether the assembly is into core or a stream. If it is a stream, then its external name is also specified; in this case the Loader provides the Assembler with the internal name, either by finding it in the user's local or the global dictionary or by creating a new stream and also making a dictionary entry for it in the local dictionary. Within the code input to the Assembler, references to streams may be made using ordinary labels. A special pseudo-operation is used to relate the labels used to the actual external stream names and also to indicate whether the external names are in the current local or the global dictionary. The Assembler builds a table of all references to streams, and the Loader fills in the internal names wherever the references occur.

In addition to executing and assembling, the Loader accepts commands to create, modify and delete dictionaries and to specify the local dictionary to be used for resolving assembled references.

Debugging

With the capabilities described above, it is possible to assemble and execute procedures. For even a minimum programming environment, it is also necessary to provide debugging tools for checking out programs. One of the objectives of the Venus microprogram design was to facilitate the implementation of instrumentation and debugging routines through interrupts. In particular, it is possible, under software control, to enable an interrupt which will occur before each instruction of a procedure is executed. The interrupt can cause another procedure to be executed without disturbing the interrupted procedure. Using this interrupt feature, an on-line, interactive debugging program was written which could be used to stop execution of a procedure at some specified 'breakpoint' or after a specified number of instructions had been executed. A dialogue with the reader then commences, in which the contents of streams and core may be examined and modified, and another breakpoint may be specified. Then control may be returned to the interrupted procedure or to the Loader. The debugging capabilities are available without making any modifications to the procedure being checked out.

THE MULTI-USER ENVIRONMENT

The Venus Multiprogramming System is an operating system which supports several jobs running concurrently. Such a system must perform the following two functions:

1. Management of resources.

   If several jobs are allowed to compete freely for resources, deadlock may result, in which each job is waiting for some other job to release a resource, and no job can continue. Very careful design of the management of resources is required to prevent this. In Venus there are three types of resources to manage: shared data streams, input/output devices, and job areas.

2. Support for the user.

   Each user in the multiprogramming system requires a running environment. This environment is an extension of the single-user environment; of the additions made, the most important is the provision of input/output functions. This is an advantage to the programmer since it frees him from the petty details of running the devices, and at the same time it allows efficient sharing of devices among users.

Semaphores

One of the most important features of the Venus microprogram, as far as the design of the multiprogramming system is concerned, is the provision of semaphores for control of shared resources and synchronization of jobs. A semaphore is a special type of data on which two operations may be performed: P and V. P is performed when a job wants to wait for something to happen: either for some piece of data to be free or for some event to occur. V is performed for an opposite reason: to free a piece of data for use by some other job or to signal that an event has occurred.

Shared Data Streams

The control of shared data streams occurs directly from the use of semaphores. An example of such a stream is a dictionary. Now that there is more than one job, it is possible that several jobs would access the dictionary at once; however, each job assumes that the dictionary is not changing while it is being used. To insure that this is true, each dictionary is protected by a semaphore; a job which wishes to access the dictionary performs a P on the semaphore before use and a V when it has finished. This means that only one job can use the dictionary at a time, and the integrity of the data is preserved.

In Venus, all data streams are available to the user as well as to the system. If semaphores are used correctly, the data is protected; however, there is some danger that the user may not use them correctly. For example, he may forget to perform the V which frees the data for use by someone else. If the stream being used is a system stream, then other jobs will be affected by his error; this is unacceptable in a multi-user environment. One way of avoiding this is to insure that access to system streams is always handled correctly. This is done in Venus by allowing the user to access system streams only through system procedures, which handle P's and V's correctly. The use of P's and V's, augmented by the system procedures, completely defines the control of shared data streams in Venus.

Synchronization

The other use of semaphores, for the synchronization of jobs, is even more important to the design of the system. Any operating system has tasks which it must perform concurrently and asynchronously if the system is to be at all efficient. At the very least, it must permit input/output to occur concurrently with ordinary processing.

In Venus, sixteen jobs are supported by the microprogram. From the point of view of the software, these jobs can all be considered to be running concurrently and asynchronously. These jobs can synchronize with each other by using P's and V's. In addition, Venus has a microprogrammed multiplex channel which runs concurrently with the jobs. The synchronization between jobs and I/O also occurs through P's and V's; the microprogrammed channel performs a V to signal completion of an I/O operation.

We felt that the design of the system would be accomplished best by allowing tasks which should logically be performed asynchronously to actually be performed asynchronously by assigning each such task to a separate job. This approach allows an elegant and simple solution to problems which become very complex if the synchronization is simulated rather than real. It also satisfies the goal of this year: It permits us to observe the effect of the new architecture on the design and production of software.

The Poller

One consequence of designing the system in this manner is that many jobs are required. This is not practical since each job requires a separate job area, and job areas are a scarce resource. A simple function called the Poller, however, permits several tasks to share the same job area. The Poller does not know which tasks are sharing the job area or even how many tasks there are. More importantly, each task is unaware that others are sharing the job area; each assumes that it is running alone. Therefore, the simplicity of the design of the system is not compromised by having tasks share job areas.

Every job area used by the system for system tasks is run by the Poller, even when only one task may be performed from that job area. This permits the redistribution of tasks to job areas later if necessary. Selection of which tasks share a job area is made on the basis of overall system efficiency.

Queues

P's and V's permit jobs to wait for or signal the occurrence of events. They do not, however, contain any information about what the event is. For example, a job may be performing a certain function every time another job requests it. It may be necessary for it to know which particular job made the request. A mechanism was needed which allows one job to store information and the same or another job to retrieve it. Queues were selected for this purpose because they supply a natural ordering for their elements. Therefore, a job can easily choose between elements on a queue to decide which function to perform next; generally, they are performed in chronological order.

Queues are a general data structure supported by the system and therefore available to users. As with dictionaries, only one user may access a queue at a time. The queues are protected by system-provided queuing functions, which use semaphores to control the access to the queue.

Management of Input/Output Devices

Another problem to be solved was the management of the various input/output devices. To do this, we had to analyze what we really wanted from the devices. The devices fall into two distinct classes: interactive devices and scarce resources.

1. Interactive Devices.

This class contains the teletypes, data sets and ARDS display. Venus is an on-line, interactive system, and every user needs access to an interactive device. However, there are not enough devices to go around; if each device were assigned exclusively to a user, fewer users could be supported. Furthermore, it would not then be natural for one user to put a message on another user's teletype or for the system to do so.

The solution chosen was to give each user a preferred device which he will ordinarily use. However, he is not constrained to do so and is unable to prevent other users from using the device. To avoid a possible chaotic situation, such a device may be owned by one user for the duration of a write followed by a read. This takes care of the standard use of interactive devices, which is the user responding to the program (with a command) when told to do so.

2. Scarce Resources.

The card reader, printer and magnetic tapes belong in this class. All users are benefited by efficient handling of these resources. If a user were permitted to gain control of one of these devices, all other users would have to wait for it until he finished with it. However, it would very likely not be used continuously, and the overall value of the device would be decreased.

Instead, each of these devices is handled by a system function, which has the responsibility of running the device as efficiently as possible. For example, whenever cards are in the card reader, they will be read; the card reader is operated independently of the users. These system functions run asynchronously with user jobs and communicate with user jobs via semaphores and queues. Each such system function defines the way in which users may access the device; for example, card input is accessed a deck at a time.

Design of Input/Output - Levels

Having decided how to perform the management of devices, it was then necessary to define a mechanism which supports this management and at the same time takes into account the physical characteristics of the devices and of the Venus machine. This mechanism has been built in levels as defined by Professor Dijkstra (3).

At the lowest (or most basic) level is the device itself together with the multiplex channel supported by the microprogram. While a device is running, the sequence of commands to control a device must be contained in core memory; the input and output buffers must also be in core. At this level, there is no control over access to a device; if one job starts using a device already running for another job, both jobs will suffer ill effects.

Controllers

The next higher level is made up of the I/O Controllers. There is one Controller per device; a Controller is aware of the physical characteristics of the device it is controlling. However, the general design of the Controllers is similar for all devices.

The primary function of a Controller is to control access to a device. It does this by means of two queues: a "request" queue and a "done" queue. Any job which wishes to use the device puts a request on the "request" queue and notifies the Controller by performing a V on the Controller's semaphore. The Controller processes the requests one by one, not starting a new request until the old one has been finished. Each time a request is finished, the Controller puts its result on the "done" queue and notifies the job which made the request (put it on the "request" queue) by performing a V on a semaphore specified as part of the request.

The request contains an indicator which selects one of the types of transfer which are supported by the device Controller. Only a small number of such types are available. This does not mean that the user has lost any important control of the device, because only minor restrictions are imposed. For example, on the teletype, a limited set of characters has been chosen to end input; at the lowest level any character can end input.

The Controllers map the type of transfer into the appropriate set of commands and buffers required to actually run the device, perform the transfer, and move the data involved in the transfer between core buffers and streams. Each Controller has a fixed area in core for its I/O commands and buffers. This means that I/O buffers have a fixed maximum size, chosen to be as small as is compatible with using the device meaningfully (since core is limited). For example, the card reader buffer has space for one card image.

Because a Controller is running asynchronously with a user requesting its device, it resides in a different job area from the user. However, most of the time the Controller is waiting for the completion of an I/O transfer. Therefore, a job area containing a single Controller would not be busy very often. Instead, all Controllers run from the same job area under the Poller.

An understanding of the effect of the Controllers is achieved by examining what a device looks like from above this level. It is a "logical" device which runs concurrently with the user. The user need merely make a request for the device when he wants to use it, without concerning himself about whether other users are also requesting the device. The transfer requested will be performed (not necessarily immediately); the user synchronizes with the completion of the transfer by defining a semaphore (made known to the Controller at the time of the request) and performing a P on it.

Requesters

The level above Controllers is made up of requesters (which make requests of the Controllers). It is at this level that the different ways in which devices are used in the system are defined.

1. Teletypes (eventually data sets and ARDS display).

In the case of teletypes, the Controllers match the way that users may access the teletypes quite closely. A Teletype Requester allows requests to be made more conveniently and also prevents user errors from causing the Controller to fail for all jobs. It builds a description of the request which the Controller understands from the somewhat freer description given by the user, puts the request on the "request" queue and notifies the Controller. It also performs the synchronization with the Controller (unless specifically asked not to), retrieves the result from the "done" queue, and supports a symbolic reference to the "job" teletype (the preferred device which the job will ordinarily use).

Since there is no synchronization necessary between the user and the Teletype Requester, the Requester runs in the user's job area and is called like any other procedure.

2. Printer.

The printer differs from the teletype in several significant ways. First, it is not an interactive device from which a user must have a response in order to continue processing. Second, it is a scarce resource which must be shared by all jobs which are running in the multiprogramming system. In addition, the Printer Controller does not match the way a user would like to access the printer as well as the Teletype Controller matches the teletype, because it may only accept a small amount of data to be printed at one time. This limitation is due to the necessity to restrict the size of the core buffer from which the actual printing takes place. If several users made requests directly to the Printer Controller to print the acceptable amount of data, then their respective sets of output would be interspersed on the paper in whatever order they were received without regard to the logical grouping of each user's data.

To avoid this situation, a second level of control has been designed for the printer. A function called the Printer Driver accepts requests to print a logical increment of data rather than just a fixed length; for example, the entire listing of an assembled procedure may be printed by one request. The data which the Printer Driver is asked to print is referred to as a print stream. The Printer Driver divides the print stream into units which can be handled by the Printer Controller and makes a succession of requests of the Controller until the whole print stream has been printed. It completely processes one print stream before it begins the next request. In this way the printed output for one Driver request will appear contiguously on the printer, and the problem of randomly interspersed data has been circumvented.

The actual printing of a print stream and the 'driving' of the Printer Controller occur asynchronously with the operation of the procedure which requests the printing. Therefore, the Printer Driver is in a different job area from the jobs which use it. Communication between the user and the Driver is accomplished through a "request" queue, on which the user places the print request. No synchronization is needed to signal the end of printing; in fact, the job which made the request may even terminate before the printing actually occurs.

There is another level above the Printer Driver called the Printer Requester. Its role is similar to that of the Teletype Requester in providing a simpler interface between the user and the printer. At the same time, it protects the system from errors the user might make if he were to deal directly with system data streams. The Printer Requester enables the user to build print streams a line at a time or to convert portions of an existing stream into a printable form. When the print stream is ready to be printed, the Printer Requester sends the request to the Printer Driver. The Printer Requester runs in the same job area as its user.

3. Card Reader.

The card reader presents an opposite problem to the printer. User's decks are made up of contiguous cards; no cards of other users are present within one user's deck. The Card Reader Driver recognizes this fact; it builds images of card decks in streams. As it builds a stream, it converts the card images into internal form.

The Card Reader Driver reads cards by sending requests to the Card Reader Controller. It runs asynchronously with and independently of the users. The Card Reader Driver reads cards whenever there are cards in the reader; in fact, the user of these cards may not even be running yet. Synchronization with the user occurs when he wishes to access the stream which contains the image of his card deck.

The Card Reader Driver occupies a separate job area from the users and also from the Controllers. (It could share a job area with the Printer Driver.) Users always access their card decks through commands to the Loader; the Loader performs the synchronization with the Card Reader Driver which prevents a user from accessing a card image stream before the cards have been read. The Loader thus performs like a Card Reader Requester.

The Executive

At the beginning of this section, three types of resources were mentioned as being in need of management: shared data streams, input/output devices, and job areas. The management of shared data streams occurs through semaphores and system functions; input/output devices are managed by the controllers and requesters; the executive manages the job areas.

Some job areas are reserved permanently for the system; the others are available to the users on a first come, first serve basis. To start a job, the user hits the break button on the teletype he wishes to have as his "job" teletype. This awakens the Controller of that device, which in turn awakens the executive. The executive then asks the user what he wants to do; to start a job, he types in START and the name of his job.

To find a job area to hold this job, the executive searches the "job use" table which tells whether each job area is in use, and if so, the name of the job which is using it. If an available job area is found, it is assigned to the user; then the job area is initialized, and the Loader started up in it. The teletype used for the command is assigned as the "job" teletype.

The executive accepts several other commands. Some of these are user commands; for example, if the user runs into trouble and wants to stop his job, he types KILL and the name of his job. The executive also performs system functions on command.

In addition to running in Command Mode (which it enters when the break button is depressed), the executive also handles special requests from jobs. For example, at normal job termination, the executive is notified to remove the job name from the "job use" table, thus freeing the job area for future use. The executive runs concurrently with users and other system functions and occupies its own job area.

SUMMARY - THE USER ENVIRONMENT

A description of the user environment provides a good summary of the operating system, since the ease with which the user can write, check out and run under the system is a measure of the success of the design. The user sees the system from two points of view. First, he must write his programs; for this he is interested in the data structures and system functions which he can use. Then he is interested in running and debugging his programs.

The programmer writing programs to be run under Venus uses streams for all storage - procedures, data and push down stacks. He never concerns himself with storage allocation - the system handles all that for him. Because streams are available by external name through the mechanism of dictionaries, users can readily share them and therefore share data and procedures. The procedures which make up the system are shared in this way; similarly, subsystems may be defined. The programmer also has available to him mechanisms, in the form of queues and semaphores, which permit him to synchronize his procedure with other users.

The programmer has very little work to do when defining the input and output part of his program. In order to do I/O on his interactive device, which will change from day to day, he refers symbolically to the "job" teletype when calling the Teletype Requester. He may also refer symbolically to the "console" teletype, should some error condition arise. The actual transfer is also described symbolically; the Teletype Requester builds a description of the transfer, notifies the Controller and performs the synchronization. The only other device with which a program deals is the printer. Here the Printer Requester builds the "print" stream for the program and sends it to be printed when the time comes.

The user runs in an on-line, interactive mode. He starts up his job by a command to the executive given on the teletype he wishes to use as his "job" teletype. He may submit card decks to be read prior to his run; when he starts to run,he may access all decks through a command to the Loader. The Loader will move the names of all his symbolic card decks to his "symbolic" dictionary; this is a new dictionary defined specifically for this purpose. In addition, the Loader will assemble all these decks automatically or on command.

The Assembler accepts input from a stream which contains the image of a card deck. The stream being assembled defines its own external name; it refers to other streams by external name. The correspondence between external and internal stream names is contained in dictionaries. The user has a wider access to dictionaries than the two levels provided in the single user environment. In addition to referring to streams whose names are in the local dictionary or in the global dictionary, reference may be made to streams in other dictionaries by mentioning the dictionary by its external name.

When the user has assembled his streams, he may execute procedures by giving a command to the Loader. The execution can occur with or without debugging; no change to the procedures being run is required. The debugging is under the control of an interactive function which runs before the execution of every instruction. This function can be used to stop execution of a procedure at a specified "breakpoint" or after a certain number of instructions have been executed. A dialogue with the user then commences in which the contents of stream or core may be displayed and modified and a new breakpoint specified. Then control may be returned to the interrupted procedure or to the Loader.

In addition to this debugging aid, the system also provides interrupt functions which run as the result of exceptional conditions; for example, a stack underflow or overflow or an illegal instruction in a procedure. These functions are interactive and make use of a subset of the debugging commands. They permit the user to discover the reason for the error, restore his data if necessary and return to the Loader.

When the user has finished running and debugging, he informs the Loader, which cleans up his job area and then notifies the executive so the job area will be made available to others. His checked out programs may be entered in the system and become accessible to others through the dictionaries.

SECTION III

MICROPROGRAMS

One of the goals of this project has been to evaluate the hazards of user microprogramming. Among other things, the user must cope with the detection and eradication of errors in the microprogram. During the course of using the Venus machine, some errors were detected in the Venus microprogram. All these errors have been corrected by making wiring changes in the read only memory.

We have been interested in analyzing why these errors were not detected during checkout. We believe this is for the following reasons:

1. Simulation. Checkout of the microprogram occurs under a simulator supplied by Interdata. This simulator contains errors and insufficiencies. One simulator error led to a wide-spread error in the microprogram.

   In addition, running under the simulator is extremely slow. This discourages complete checkout, especially when complex interrelations between parts of the microprogram are involved. The ideal way to check out, which is impractical under the current simulator, would be to actually write the software which the microprogram is supposed to support and then attempt to run it.

2. Complexity. The Venus microprogram supports very sophisticated and complex functions which were designed with extreme care. However, the read only memory would not be large enough to contain the microprogram if it were coded in a clean and straightforward manner. The additional coding complexity required to reduce the size of the microprogram led to errors.

SECTION IV

HARDWARE

MAINTENANCE

Intermittent and chronic hardware troubles have occurred, but by the end of the year all major difficulties had been eliminated. Problems with the memory and disk were finally attributed to overheating in the equipment cabinet, and the installation of a second set of fans alleviated the problem. The Motorola printer had difficulty keeping its heads in position and persistent servicing finally kept the malfunction under control. An improvement was installed by Interdata on the card reader which has made it more reliable.

Effort has been expended on improving the equipment checkout programs which we have written for determining equipment failures. These programs have been successful in detecting errors not revealed by Interdata testing, thereby saving considerable time in correcting the errors.

NEW HARDWARE

Data Sets and Adapters

Two data sets and adapters were installed to allow use of remote teletypes. The teletypes in the facility may be directly connected to the computer or connected to the data sets by a switch which was designed and built at MITRE. This capability greatly facilitated checkout of remote teletype operation.

Magnetic Tapes Drives

Two Peripheral Equipment Corporation tape drives were installed. Equipment checkout programs, written to test them for acceptance, detected a hardware design error which was corrected by Interdata. In May the tape drives were accepted and are an important adjunct to the facility. They are IBM-compatible and can also serve as a means of transporting data to and from the IBM 360/50.

Selector Channel

At the end of the year a second selector channel was installed. This allows the other selector channel to be dedicated to the Data

Disk, which is heavily used to support paging of virtual memories in the Venus System. The tape drives are connected to the new selector channel, and eventually a large disk will also use that channel.

Real-Time Clock

A second real-time clock was installed in June. This clock will be available for user applications such as time interval interrupts. The original clock is dedicated to supporting time-sharing operations and accounting.

SECTION V

OTHER ACTIVITIES

In October, a paper entitled "The Venus Multiprogramming System" was presented at the Second Annual Microprogramming Workshop in Phoenix, Arizona. The text of this paper was later published (1).

Also in October, a member of the staff attended the Second Symposium on Operating System Principles at Princeton University, Princeton, New Jersey. Two papers, one on the MI-3 Assembler and a microprogramming bibliography (4), were published in October.

Two briefings on Microprogramming were presented in January, one to the SAMSO Technical Panel and one to ESD.

In February, a paper describing the Adapter from the 103 Data Set to the ASR33 Teletype was published.

A member of the staff was a panelist in the Microprogramming Session of the Spring Joint Computer Conference, held in Atlantic City in May. Also in May, a paper describing the Venus machine (2) was published.

In June, the Venus Multiprogramming paper was again presented, this time at the Grenoble Workshop on Microprogramming in Grenoble, France.

REFERENCES

1. B. J. Huberman and R. G. Curtis, The MITRE Corporation, "The Venus Multiprogramming System," M69-69. Contract F19(628)-68-C-0365, Bedford, Mass. November 1969.

2. B. J. Huberman, The MITRE Corporation, "Principles of Operation of the Venus Microprogram," MTR-1843. ESD-TR-70-198, Contract F19(628)-68-C-0365, Bedford, Mass. July 1970.

3. Professor E. Dijkstra, "The Structure of the 'THE' — Multiprogramming System," Communications of the ACM, Vol. 11, No. 5, May 1968.

4. The MITRE Corporation, "Annotated Microprogramming Bibliography," M69-65. ESD-TR-70-204, Contract F19(628)-68-C-0365, Bedford, Mass. July 1970.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* The MITRE Corporation Bedford, Massachusetts	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

THE VENUS MULTIPROGRAMMING SYSTEM — YEAR END SUMMARY

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

N/A

5. AUTHOR(S) *(First name, middle initial, last name)*

Barbara H. Liskov

6. REPORT DATE JANUARY 1971	7a. TOTAL NO. OF PAGES 25	7b. NO. OF REFS 4

8a. CONTRACT OR GRANT NO. F19(628)-68-C-0365	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-70-408
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* MTR-2004, Rev. 1
d.	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES N/A	12. SPONSORING MILITARY ACTIVITY Electronic Systems Division, Air Force Systems Command, L. G. Hanscom Field, Bedford, Massachusetts

13. ABSTRACT

The Venus Multiprogramming System is an on-line, interactive system in which several users may run concurrently. It is built on the Interdata 3 from a combination of microprograms and software. This document describes the activity of Project 700A, C^3 Computer System Organization, for fiscal year 1970. The work performed consists of the design and implementation of the software part of Venus. The document gives a chronological account of the development of the software, with particular emphasis on the general principles incorporated in the system and the design considerations which led to these principles.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
COMPUTER PROGRAMMING						
COMPUTER PROGRAMS						
COMPUTER SOFTWARE						
COMPUTER SYSTEMS PROGRAMS						
MICROPROGRAMMING						
PROGRAMMING TECHNIQUES						
SOFTWARE (COMPUTERS)						
VENUS MICROPROGRAM						